Introducción

Arquitectura de Computadores – IIC2343

El hardware de todo computador ejecuta cuatro funciones básicas:

- ingresar o recibir (*input* de) datos
- emitir o enviar (*output* de) datos
- procesar datos
- almacenar datos

Este curso: cómo se llevan a cabo estas funciones

Así, las cinco componentes clásicas de un computador —pasado o presente— independientes de la tecnología de hardware:

- input
- output
- memoria
- *datapath* (componente que ejecuta las operaciones aritméticas)
- control

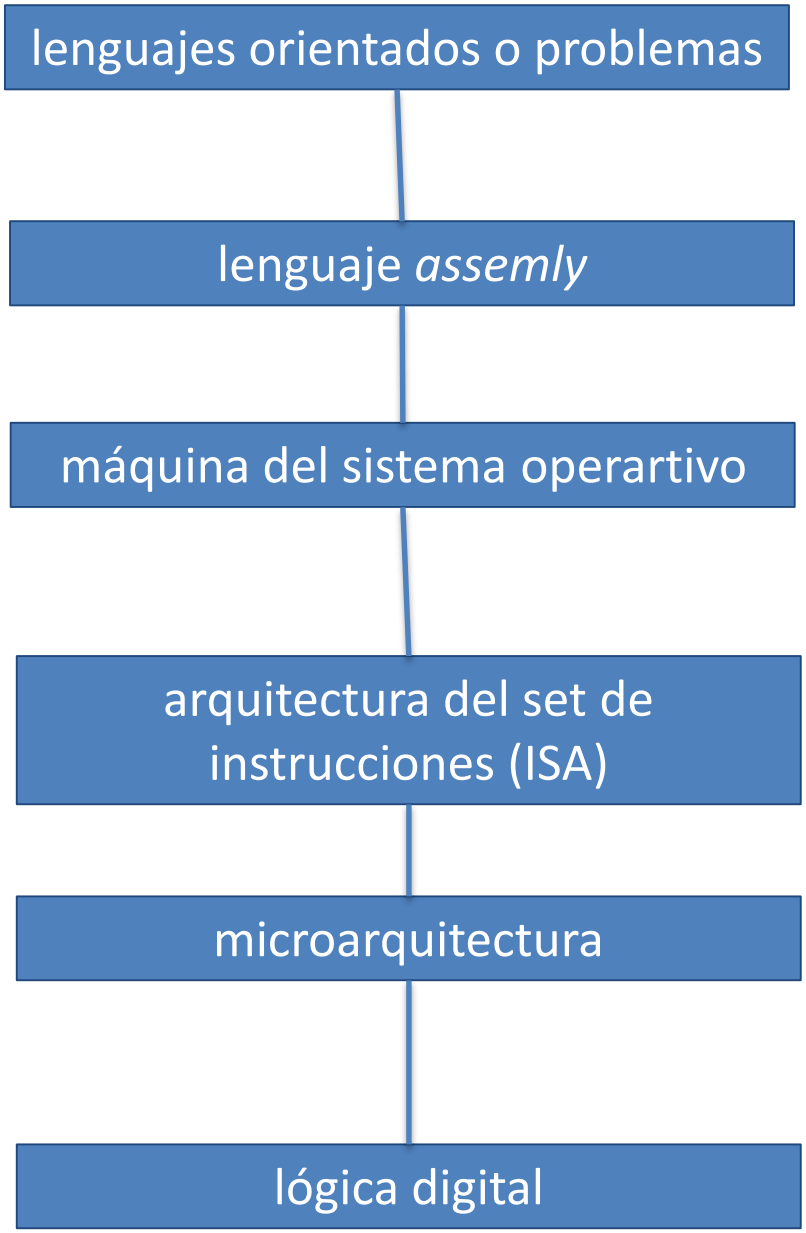
Una aplicación típica:

- millones de líneas de código
- + librerías de software que implementan funciones complejas

Pero … el hardware de un computador solo puede ejecutar instrucciones de “bajo nivel” muy simples

Varias capas de software organizadas jerárquicamente interpretan o traducen las operaciones de “alto nivel” a las instrucciones simples del computador:

- las capas de más “arriba”: aplicaciones, directamente disponibles para nosotros los usarios
- **las capas intermedias: software de sistemas**, en particular, sistemas operativos, compiladores, *loaders*, y *assemblers*
- la capa de más “abajo”: hardware



5) lenguajes de alto nivel, traducidos por **compiladores**, o interpretados

4) forma simbólica de los lenguajes de más abajo, traducida por el **assembler**

3) instrucciones adicionales, otra organización de memoria, capacidad de ejecución concurrente → interpretado por un programa llamado el sistema operativo

2) instrucciones ejecutadas por el microprograma o circuitos del hardware

1) 32 registros + **ALU** → **datapath**, **microprogramado** o controlado por hardware

0) compuertas (gates), álgebra de Boole, registros, el motor de computación

Sistema operativo:

- p.ej., Linux, iOS, Windows

… es la interfaz entre los programas de los usuarios y el hardware

… proporciona servicios y funciones de supervisión:

- manejo de operaciones básicas de i/o
- asignación de almacenamiento y memoria
- compartimiento protegido del computador entre muchas aplicaciones que lo usan simultáneamente

Compilador

… traduce un programa escrito en un lenguaje de alto nivel —C, C++, Java, Visual Basic— a instrucciones que el hardware puede ejecutar:

- la traducción es un proceso complejo

Para “hablarle” al hardware electrónico, hay que enviarle señales eléctricas:

- las más fáciles son *on* y *off*

  … ➔ el alfabeto del computador tiene solo dos letras, cuyos símbolos son los números 0 y 1

  … ➔ cada letra es un **dígito binario**, o **bit**

Los comandos —*instrucciones de máquina*— son secuencias de bits que el computador entiende y obedece (y que podemos interpretar como números en base 2, o binarios, como ya veremos):

- p.ej., `1000110010100000` le dice al computador que sume dos números

Los primeros programadores se comunicaban con los computadores usando estos números binarios

Luego, los programadores inventaron notaciones más cercanas a nuestra forma de pensar:

- primero, estas notaciones eran traducidas a la notación binaria a mano

… y programas para traducir de esta notación simbólica a la binaria — el ensamblador o *assembler*:

- … es decir, usaron el computador para que los ayudara a programar el propio computador

P.ej., si el programador escribe

`add A,B` —representación simbólica de una instrucción de máquina

… entonces el assembler traduce a

`1000110010100000` —representación binaria de la instrucción de máquina

El lenguaje simbólico se llama lenguaje de ensamble, o **assembly**

El lenguaje binario, que es el que el computador entiende realmente, es el **lenguaje de máquina**

El lenguaje assembly aún está lejos de las notaciones que usa un científico o una ingeniera en su actividad profesional

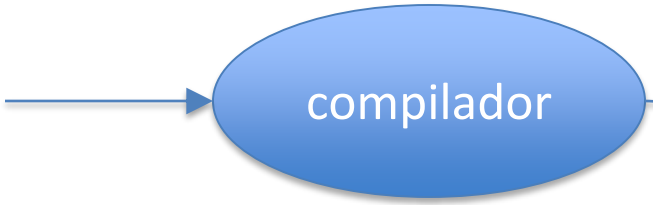
… de modo que los programadores inventaron lenguajes de programación de (más) alto nivel

… y compiladores que traducen programas escritos en estos lenguajes a instrucciones:

- se dieron cuenta de que se puede escribir un programa para traducir un lenguaje más poderoso a instrucciones de máquina

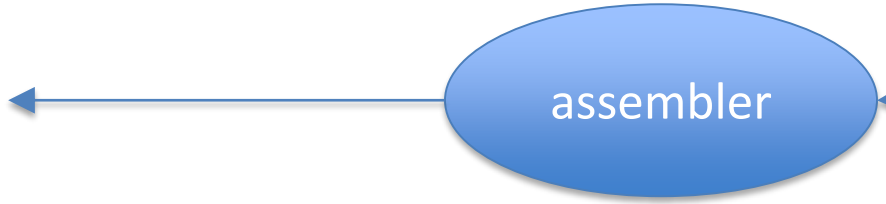
```
swap(int v[], int k):
    int temp
    temp = v[k]
    v[k] = v[k+1]
    v[k+1] = temp
```

programa
en lenguaje
de alto nivel



programa
en lenguaje
assembly

```
swap:
    mult   $2, $5,4
    add $2, $4,$2
    lw  $15,    0($2)
    lw  $16,    4($2)
    sw  $16,    0($2)
    sw  $15,    4($2)
    jr  $31
```



programa
en lenguaje de
máquina binario

```
0000 … 1000
0000 … 0001
1000 … 0000
1000 … 0100
1010 … 0000
1010 … 0100
0000 … 1000
```

Beneficios de los lenguajes de programación de alto nivel:

- permiten al programador pensar en un lenguaje más natural, usando palabras en inglés (**`if`**, **`while`**) y notación algebraica (**`b*b - 4*a*c`**)
- mejora la productividad del programador —toma menos tiempo desarrollar programas cuando los escribimos en lenguajes que requieren menos líneas para expresar una idea
- los programas son independientes del computador en el que fueron desarrollados —los compiladores y assemblers pueden traducir los programas escritos en lenguajes de alto nivel a las instrucciones binarias de cualquier computador